SANDIA REPORT

SAND2005-6934
Unlimited Release
Printed September 2005

Visualizing Higher Order Finite Elements: FY05 Yearly Report

D. C. Thompson and P. P. Pébay

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2005-6934xx
Unlimited Release
Printed September 2005

Visualizing Higher Order Finite Elements: FY05 Yearly Report

D. C. Thompson
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94550, U.S.A.
dcthomp@sandia.gov

P. P. Pébay
Sandia National Laboratories
M.S. 9051, P.O. Box 969
Livermore, CA 94550, U.S.A.
pppebay@ca.sandia.gov

Abstract

This report contains an algorithm for decomposing higher-order finite elements into regions appropriate for isosurfacing and proves the conditions under which the algorithm will terminate. Finite elements are used to create piecewise polynomial approximants to the solution of partial differential equations for which no analytical solution exists. These polynomials represent fields such as pressure, stress, and momentim. In the past, these polynomials have been linear in each parametric coordinate. Each polynomial coefficient must be uniquely determined by a simulation, and these coefficients are called degrees of freedom. When there are not enough degrees of freedom, simulations will typically fail to produce a valid approximation to the solution. Recent work has shown that increasing the number of degrees of freedom by increasing the order of the polynomial approximation (instead of increasing the number of finite elements, each of which has its own set of coefficients) can allow some types of simulations to produce a valid approximation with many fewer degrees of freedom than increasing the number of finite elements alone. However, once the simulation has determined the values of all the coefficients in a higher-order approximant, tools do not exist for visual inspection of the solution.

This report focuses on a technique for the visual inspection of higher-order finite element simulation results based on decomposing each finite element into simplicial regions where existing visualization algorithms such as isosurfacing will work. The requirements of the isosurfacing algorithm are enumerated and related to the places where the partial derivatives of the polynomial become zero. The original isosurfacing algorithm is then applied to each of these regions in turn.

Acknowledgement

The authors would like to thank David Day and Louis Romero for their insight into polynomial system solvers and the LDRD Senior Council for the opportunity to pursue this research. The authors were supported by the United States Department of Energy, Office of Defense Programs by the Labratory Directed Research and Development Senior Council, project 90499. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

Contents

Figures

This page left intentionally blank

Visualizing Higher Order Finite Elements: FY05 Yearly Report

1 Introduction

1.1 Motivation

Sandia's role as a stockpile steward depends on its ability to explore nuclear weapon performance numerically, rather than experimentally. This numerical simulation is most often accomplished through finite element analysis, a technique under constant development over the last 3 to 4 decades. Recent advances in finite element methods increase both the hierarchical (h) and polynomial (p) level of detail – or *degrees of freedom* – during a simulation. Finite element methods have advanced significantly since their conception several hundred years ago. Much more recently, a class of techniques known as hp-adaptive methods have been developed in an effort to converge to a solution more quickly than previously possible. Finite element solvers that incorporate hp-adaptivity are quickly becoming popular since they often converge to a solution with fewer total degrees of freedom than hierarchical adaptivity alone. Among others, the SIERRA team is implementing higher order polynomial finite element solvers.

Currently, there is no way to visualize the solutions of simulations with cubic or higher order elements, which prevents analysts from exploiting using high order simulations. This is a critical issue, and as long as it will remain unsolved, it is unlikely that higher order finite element simulations will be of any production-level use. For instance, although tools such as ParaView and Ensight can currently render finite elements of degree 2, they do not always do so correctly. Figure 1, left, shows the same scalar field over a quadratic element's boundary rendered with ParaView's standard technique (back) and after our refinement filter has been applied (front). Figure 1, right, is the same but with volume rendering instead of surface rendering.

1.2 Context: the (nonlinear) finite element method

Here we briefly review the finite element method to develop notation used throughout the paper. Recall that the finite element method approximates the solution, $f : \Omega \mapsto \mathbb{R}$, of some differential equation as a set of piecewise functions over the problem domain, $\Omega \subset \mathbb{R}^d$. Although Ω may be any general d-dimensional domain, we'll assume it is 3-dimensional. The fact that we have a piecewise approximant divides Ω into subdomains $\Omega_e \subseteq \Omega$, $e \in E_\Omega$ that form a closed cover of Ω. Each Ω_e is itself a closed set of points with its own approximating function f_e. Furthermore, we require that each Ω_e be parameterized so that

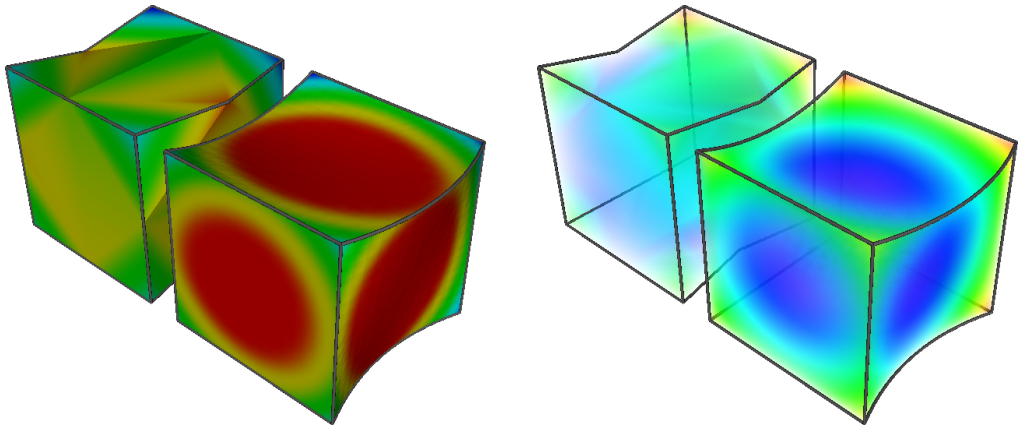

Figure 1. Surface and volume renderings of the same quadratic hexahedral finite element.

the approximating functions can be written as polynomials in the parameters:

$$\Phi_e(\vec{r}) = A_{0,0,0} + A_{1,0,0}r + A_{0,1,0}s + A_{0,0,1}t + A_{1,1,1}rst + \cdots$$

These coefficients, $A_{i,j,k}$, are known as *degrees of freedom* (DOFs). Each one corresponds to a particular modal shape, and sets of modal shapes can be grouped together into nodes by the regions of parameter-space over which they have an effect (a given corner, edge, face, or the interior volume).

Because each Ω_e is parameterized, there is also a map from parametric coordinates $\vec{r} = (r,s,t)$ to geometric coordinates $\vec{x} = (x,y,z)$:

$$\Xi_e(\vec{r}) = B_{0,0,0} + B_{1,0,0}r + B_{0,1,0}s + B_{0,0,1}t + B_{1,1,1}rst + \cdots$$

where $B_{i,j,k} \in \mathbb{R}^3$. So, the approximate solution to the differential equation may be written in terms of parametric coordinates (as Φ_e) or in terms of geometric coordinates:

$$f_e(\vec{x}) = \Phi_e \circ \Xi_e^{-1}$$

where we assume that Ξ_e is invertible for all points of Ω_e, as schematically represented in Figure 2. A global approximant $f(\vec{x}$ can then be constructed from the piecewise elemental approximants. This leaves only the matter of what to do where Ω_e and $\Omega_{j,j\neq e}$ intersect. Usually, these subdomains intersect over $(d-1)$-dimensional or lower regions (2-dimensional faces, 1-dimensional edges, and "0"-dimensional vertices in our case). In these regions, Φ is not well-defined since Φ_e and Φ_j may disagree. Usually, the finite element method constrains Φ_e and Φ_j to be identical, however some methods such as the discontinuous Galerkin method do not require this and subsequently have no valid approximant in these regions.

For a given decomposition of Ω, the finite element method may not converge to the correct (or indeed, any) solution. When Φ_e and Ξ_e are trilinear polynomials for all i, a technique

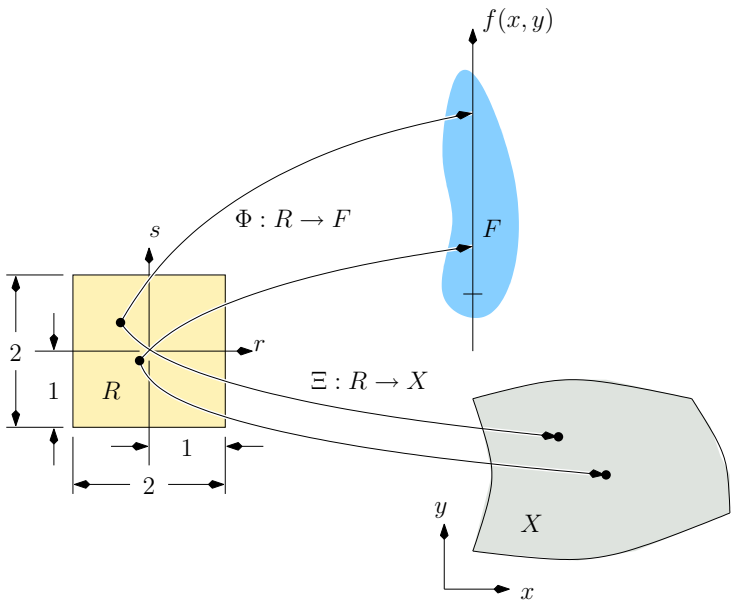

Figure 2. The 2 maps involved in the finite element approximation of the solution.

called *h*-adaptation is often used to force convergence and/or increase solution accuracy. In this technique, some subdomains $E \subseteq E_\Omega$ are replaced with a finer subdivision E' such that $\bigcup_{e \in E} \Omega_e = \bigcup_{e \in E'} \Omega_e$ but $|E'| > |E|$[1].

Similarly, *p*-adaptation is the technique of increasing the order of polynomials Φ and/or Ξ rather than the number of finite elements. *hp*-adaptation is then some combination of *h*- and *p*-adaptation over Ω. Usually, *h*- and *p*-adaptation occur in mutually exclusive subdomains of Ω, but this is not a strict requirement. Note that while *h*-adaptation generates a finer approximation of both Φ and Ξ wherever it occurs, *p*-adaptation can selectively refine individual fields.

In the end, the finite element method provides an approximation to Φ by solving a collection of equations for coefficients (A and B in the examples above). Our task is then to charactize the maps Φ and Ξ in a way that aids human understanding of the solution.

1.3 Statement of the Problem

Higher order surface rendering is probably the most used visualization technique, which we have already addressed with a brute-force technique that scales with the size of the

[1] In temporal simulations, we do allow $|E'| < |E|$ in regions where Ω has been adapted to some time-transient phenomenon.

model [2]. Arguably, the next most useful visualization tool is isocontouring. For higher order finite element solution fields, this pertains to a branch of mathematics completely different from that used to compute them. In fact, it involves solving a problem proposed by D. HILBERT as an

> [...] investigation as to the number, form, and position of the sheets of an algebraic surface in space[2].

Obviously, we are not going to analytically solve this problem that has stymied mathematicians for more than a hundred years, and still does. However, even the most basic tools for understanding higher order simulation results require us to confront this issue, because without it we cannot even know the range of values a field takes over an element. Therefore, we must devise numerical approximations to this problem, and this is the goal of this research. This has lead us to develop a method for adapting existing visualization techniques to higher order finite element data.

In fact, HILBERT's 16th Problem is related to typical visualizations, not only isosurfacing, such as cutting, clipping and volume rendering, because all of these operations involve the manipulation of an algebraic field Φ defined over some region of space R. The crux of the problem is to characterize this map Φ.

One way to completely characterize Φ is to use existing techniques such as linear isosurfacing, volume rendering, *etc*. on subregions $R_i \subset R$ where the assumptions of these methods hold. This involves the identification of the set $C \subset R$ of critical values of Φ, from which R_i may be derived, from which a new tessellation of R may be computed. Then each of these visualization techniques may be applied to the new tessellation. See Figure 3. We

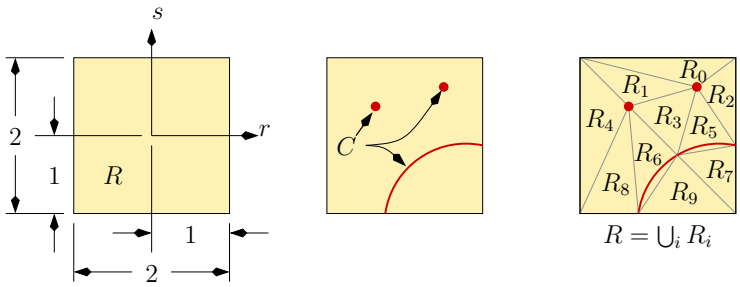

Figure 3. Decomposing a parametric domain into regions where linear assumptions hold.

have found several methods that exist to identify the points of C when they are isolated, and we have implemented one (homotopy) [8]. We also have devised [3], implemented,

[2] D. HILBERT's 16th Problem, International Congress of Mathematicians, Paris, 1900

and validated [9, 4, 7] an adaptive *streaming* tessellation scheme that is well suited to the last step. The fact that it is a streaming algorithm makes it:

1. embarrassingly parallel on clustered architectures, and
2. potentially portable to streaming FPGA/GPU-type hardware.

Therefore, what remains is the middle step of computing the R_i from C, which is complex but where methods exist. However, when some critical points are not isolated, none of the existing critical point detection techniques work and the problem remains largely open.

The goal of this LDRD-funded research was to propose one numerical solution to this problem. In this report, we present our approach and in illustrate it with isocontouring.

2 Isocontouring of Higher Order Elements

This section discusses how we solved the problem of higher order elements isocontouring. We present a detailed treatment of how critical points may be used to partition a finite element into regions where linear isosurface extraction assumptions are valid. It is very important to be aware that just because we are partitioning Ω_e into regions where the assumptions of a linear algorithm are valid, that does *not* mean that we intend to approximate the higher order interpolant with a linear interpolant in these regions! In fact, our goal once we have identified the partition is to use the higher order interpolant directly. We will show how important this distinction is for isosurfacing later in the paper.

2.1 Partitioning for Isocontouring

Here are the conditions that linear tetrahedral isosurfacing algorithms require:

(C1′) each tetrahedron edge intersects an isocontour of a particular value at most once,

(C2′) no isocontour intersects a tetrahedron face without intersecting at least two edges of the face,

(C3′) no isocontour is completely contained within a single tetrahedron, and

(C4′) the map from parametric to geometric coordinates must be bijective.

In order to adapt this algorithm to handle higher order cells of any shape and polynomial order, we will decompose Ω_e into a simplicial complex, each tetrahedron of which must satisfy the criteria above. We now translate these criteria into more precise requirements on Φ_e:

(C1) Φ_e has no interior extrema along any tetrahedral edge. We may specify an edge as a linear relationship among parametric coordinates so that, if the dependency is in terms of r, this constraint may be written $g'(r) = (\frac{\partial \Phi_e}{\partial r} + a\frac{\partial \Phi_e}{\partial s} + b\frac{\partial \Phi_e}{\partial t})_{(r,s(r),t(r))} \neq 0$ over some interval $]r_0, r_1[\subset \mathbb{R}$, where a and b are real constants. Note that one can have $g'(r) = 0$ (i.e., an extrema on the edge) even though $\nabla\Phi_e(r, s(r), t(r)) \neq 0$, which means that a critical point of g is *not* necessarily a critical point of Φ_e – although the converse is true. Also note that $(C1)$ is slightly stronger than (C1′), as the condition $g'(r) \neq 0$ is *sufficient* to avoiding multiple isocontour-edge intersections – but not necessary: just think of $r \mapsto r^3$ in 0.

(C2) Φ_e has no interior extrema over any tetrahedral face. We may specify a face as a linear relationship among parametric coordinates so that this constraint may be written, if the dependency is that of t in terms of r and s, $(\frac{\partial \Phi_e}{\partial r} + a\frac{\partial \Phi_e}{\partial t}, \frac{\partial \Phi_e}{\partial r} + b\frac{\partial \Phi_e}{\partial t})_{(r,s,t(r,s))} \neq 0$ over some open domain $U_i \subset \mathbb{R}^2$, where a and b are real constants. Once again, this expression can vanish (i.e., both components are 0) even at a point where $\nabla\Phi_e$ does not. Also, (C2) is slightly stronger than (C2′), because of saddle points (not to mention degenerate critical points),

(C3) Note that for a component of an isocontour to be completely contained in some tetrahedron, a critical point of the differentiable Φ_e must exist somewhere in the element. Thus, we must insure that all extrema of Φ_e must occur at vertices of the simplicial decomposition of Ω_e. Indeed, (C3) and (C3′) are equivalent.

(C4) We may also state (C4′) as $\forall \vec{x} \in \Xi_e(\Omega_e)$, $\exists! \, \vec{r} \in \Omega_e$ such that $\Xi_e(\vec{r}) = \vec{x}$.

So, for isocontouring, all of the differences between the linear and higher order implementation can be attributed to critical points. In fact, it is straightforward to see that similar requirements arise in the case of surface rendering.

As we noted then, higher order finite elements that have non-simplicial domains (such as hexahedra, pyramids, etc.) will have to be decomposed into tetrahedra $\mathcal{T}$. However, these tetrahedra must additionally meet the requirements (C1) to (C4). We can start by inserting corner points C_e and any points where $\nabla\Phi_e = 0$ into P_e. Although any $\mathcal{T}(P_e)$ will clearly meet (C3), the restriction of Φ_e to the edges and faces of $\mathcal{T}$ must be considered if we are to satisfy (C1) and (C2). This is because a finite element's domain Ω_e is a closed set and solving $\nabla\Phi_e = 0$ only provides critical points on the interior of Ω_e, which is an open set. In order to find maxima and minima on $\partial\Omega_e$, we must find critical points on the restriction of Φ_e to each bounding surface. But again, these critical points will only be for the open interior of each surface, so the restriction of Φ_e to each curve bounding each surface must be considered. These critical points are only for the open interior of each curve, so the values at curve endpoints – the topological corners of Ω_e – must be considered.

So, let's examine how $\mathcal{T}$ might be constructed to accomodate (C1) and (C2); although it is possible to perform a series of edge flips on the tessellation in Figure 4 so that the connec-

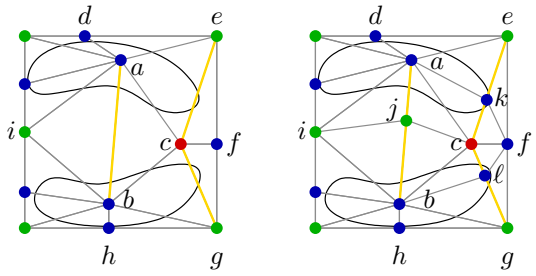

Figure 4. Critical points are shown as blue dots (maxima), red dots (saddles), and green dots (minima). Incorrect tessellation of critical points can yield simplices whose edges intersect an isosurface more than once. These problem edges are shown in yellow.

tivity is correct, it is unclear whether this holds in general – especially in three dimensions[3]. Rather than attempting to identify a set of "problem" edges and prove that they may always be flipped into a satisfactory configuration, we simply introduce a new vertex along each internal edge that has a critical point on the restriction of Φ to its domain. The vertex is then connected to each node in its star. Any new edges created by this operation must be examined for critical points of Φ restricted to their domain. This makes the algorithm recursive, but it must terminate when critical points are isolated.Although the introduction of these additional vertices partitions higher order elements into a larger number of regions, there is no constraint on the initial tessellation of the corner and critical points – which significantly reduces the amount of work.

In three dimensions, the algorithm must be modified because the introduction of a new vertex results in the creation of triangles as well as edges. Now when a new vertex is inserted, any triangles whose planes have not been previously searched for critical points must be inspected (in addition to new edges). We are spared some work since some triangles will already have been searched.

When a partition of a cell meets all the criteria of (C1) through (C4), we say that the partition is Φ-*compatible*.

2.2 Isocontouring Implementation

Although 2.1 presents a good algorithm, it is not necessarily an efficient implementation. Particularly, it makes no guarantees that shared element boundaries will be tessellated identically by all finite elements on that boundary. This prohibits a streaming implementation. It also fails to address the question of how the decomposition of each element should be

[3]Note that random triangulations may always be converted to Voronoi triangulations by flips in 2-D, but **not** in 3-D, which is one reason we are skeptical of the flipping approach.

stored; if each element is responsible for storing its own decomposition, we may either end up duplicating work computing shared boundaries or maintaining large amounts of state information as boundary constraints are propagated through the mesh.

In order to avoid these problems, we break the algorithm into several passes.

PARTITION-MESH(M,κ)
1 $C \leftarrow$ DOF-CRITICALITIES(M,κ)
2 $T_0 \leftarrow$ TRIANGULATE-BOUNDARIES(M,C)
3 $(T_1,S) \leftarrow$ TETRAHEDRALIZE-INTERIOR(M,T_0,C)
4 CORRECT-TOPOLOGY(M,κ,S,T_1)
5 **return** T_1

The first step in the process is the location of critical points interior to each finite element and each finite element's open boundaries. Because finding critical points is a time-consuming process, we do not wish to process the same shared edge or face twice. This extra work can be avoided by storing critical points indexed by the DOF with which they are associated – critical points on a face are stored with the index used to retrieve the coefficients for that face's degrees of freedom, and likewise for edges.

The technicalities involved with finding these critical points arise from the fact that

$$\begin{array}{rccc} \Phi_e: & R \subset \mathbb{R}^3 & \longrightarrow & \mathbb{R} \\ & (r,s,t)^T & \mapsto & \Phi_e(r,s,t). \end{array}$$

is defined over a closed domain R, which involves looking for critical in both the interior of R and on its closure, that can itself be decomposed in open faces, open edges, and corners. Now, a planar face that passes through the point (c_x,c_y,c_z) and has basis $\{(a_x,a_y,a_z),(b_x,b_y,b_z)\}$ can be parameterized with two variables, u and v:

$$\begin{array}{rccc} \eta: & U \subset \mathbb{R}^2 & \longrightarrow & \mathbb{R}^3 \\ & \begin{pmatrix} u \\ v \end{pmatrix} & \mapsto & \begin{pmatrix} a_x u + b_x v + c_x \\ a_y u + b_y v + c_y \\ a_z u + b_z v + c_z \end{pmatrix}, \end{array}$$

from where we find the restriction of Φ_e to the aforementioned plane:

$$G_e = \Phi_e \circ \eta : U \longrightarrow \mathbb{R}$$

Now, consider an arbitrary point (r_0,s_0,t_0) in $\mathbb{R}^3$. We then have $\mathrm{d}\Phi_e(r_0,s_0,t_0) \in \mathcal{L}\left(\mathbb{R}^3,\mathbb{R}\right)$, with

$$\mathrm{d}\Phi_e(r_0,s_0,t_0) = \nabla\Phi_e(r_0,s_0,t_0)\cdot(\mathrm{d}r,\mathrm{d}s,\mathrm{d}t),$$

where $\cdot$ denotes the usual inner product of $\mathbb{R}^3$. On the other hand, at any arbitray point

(u_0, v_0) in $\mathbb{R}^2$, $\mathrm{d}\eta\,(u_0, v_0) \in \mathcal{L}\left(\mathbb{R}^2, \mathbb{R}^3\right)$ with

$$\begin{aligned}\mathrm{d}\eta\,(u_0, v_0) &= \left.\begin{pmatrix} \frac{\partial \eta_0}{\partial u} & \frac{\partial \eta_0}{\partial v} \\ \frac{\partial \eta_1}{\partial u} & \frac{\partial \eta_1}{\partial v} \\ \frac{\partial \eta_2}{\partial u} & \frac{\partial \eta_2}{\partial v} \end{pmatrix}\right|_{(u_0,v_0)} \begin{pmatrix} du \\ dv \end{pmatrix} \\ &= \begin{pmatrix} a_x & b_x \\ a_y & b_y \\ a_z & b_z \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix}\end{aligned}$$

and therefore, $\mathrm{d}G_e\,(u_0, v_0) \in \mathcal{L}\left(\mathbb{R}^2, \mathbb{R}\right)$ with

$$\begin{aligned}\mathrm{d}G_e\,(u_0, v_0) &= \mathrm{d}\Phi_e\,(\eta\,(u_0, v_0)) \circ \mathrm{d}\eta\,(u_0, v_0) \\ &= \left.\left(\frac{\partial \Phi_e}{\partial r} \frac{\partial \Phi_e}{\partial s} \frac{\partial \Phi_e}{\partial t}\right)\right|_{\eta(u_0,v_0)} \begin{pmatrix} a_x & b_x \\ a_y & b_y \\ a_z & b_z \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix}\end{aligned}$$

Finally, critical points of the face restriction occur when $\mathrm{d}G_e(u_0, v_0) = 0$, which may also be expressed as

$$\begin{aligned} a_x \frac{\partial \Phi_e}{\partial r}\,(\eta\,(u_0, v_0)) + a_y \frac{\partial \Phi_e}{\partial s}\,(\eta\,(u_0, v_0)) + a_z \frac{\partial \Phi_e}{\partial t}\,(\eta\,(u_0, v_0)) &= 0 \\ b_x \frac{\partial \Phi_e}{\partial r}\,(\eta\,(u_0, v_0)) + b_y \frac{\partial \Phi_e}{\partial s}\,(\eta\,(u_0, v_0)) + b_z \frac{\partial \Phi_e}{\partial t}\,(\eta\,(u_0, v_0)) &= 0. \end{aligned}$$

Regarding the critical points of edge restrictions, similar calculations must be done, that will be left to the reader as an exercise.

DOF-CRITICALITIES(M)

```
1  for e ← |M|
2      do Find c. p. of Φ_e in Ω_e
3         Store c.p. indexed by volume DOF node.
4         for i ∈ BDY^2(Ω_e)
5             do if ∇Φ_{e|f_i^2} = 0 not marked,
6                    then Find c.p. of Φ_{e|f_i^2}
7                         Store c.p. of Φ_{e|f_i^2} in C_i
8                         Mark Φ_{e|f_i^2} as done
9         for i ∈ BDY^1(Ω_e)
10            do if ∇Φ_{e|f_i^1} = 0 not marked,
11                   then Find c.p. of Φ_{e|f_i^1}
12                        Store c.p. of Φ_{e|f_i^1} in C_i
13                        Mark Φ_{e|f_i^1} as done
14 return C
```

Once the critical points have been located, we triangulate two-dimensional boundaries of all cells. This ensures that any volumetric cells that reference a particular face use the same triangulation – otherwise our model would have cracks along element boundaries.

TRIANGULATE-BOUNDARIES(M,C)

```
1  T_0 ← ∅
2  for i ← each 2-boundary of every finite element
3       do P ← corner points of face i
             ⋃ isolated c.p. of all bounding edges of face f_i
4          if |C_i| > 0
5            then c ← C_{i,0}
6            else c ← FACE-CENTER(BDY_i^2(Ω_e))
7          T ← STAR^2(c,P)
8          for c ∈ {C_i \ C_{i,0}}
9               do Find t ∈ T s. t. c ∈ t
10                 Remove t from T
11                 Subdivide t into 2 or 3 triangles t_k
12                 Insert t_k into T and recursively call TRIANGULATE-BOUNDARIES
13         Insert T into T_0
14 return T_0
```

This algorithm is justified by the fact that

Proposition 2.1. *If, for all e in M, Φ_e does not have nonisolated critical points over Ω_e, nor does any of its restrictions to the faces of e, then Algorithm* TRIANGULATE-BOUNDARIES *terminates.*

Proof. To establish this result, it is sufficient to make sure the algorithm terminates, starting from any arbitray face of an arbitray element in M. So, let f_i be one of the faces of an arbitrary $e \in M$, and we then shall prove that TRIANGULATE-BOUNDARIES$(\{f_i\}, C_{|f_i})$ terminates.

First, remark that if the restriction $\Phi_{e|f_{ij}}$ of Φ_e to one edge f_{ij} of f_i has a nonisolated critical point, then this means that the derivative of $\Phi_{e|f_{ij}}$ vanishes along a nonempty open segment of f_{ij}, and therefore has an infinity of zeros. Because this derivative is itself a univariate polynomial function, it can thus only be zero everywhere, and thus $\Phi_{e|f_{ij}}$ is constant along the edge. Therefore, the only case when nonisolated critical points along a bounding edge of f_i arises is when the interpolant is constant along that edge, and therefore no other points than its endpoints are contained in P. P is indeed a finite set, as polynomials can only have a finite number of isolated critical points.

Now assume the restriction $\Phi_{e|f_i}$ of Φ_e to the interior of f_i has $n \in \mathbb{N}^*$ critical points. The innermost loop of Algorithm TRIANGULATE-BOUNDARIES will insert these n points, and

yield a triangulation of f_i in $N \in \mathbb{N}^*$ triangles $t_{i,k}$, such that

$$\begin{cases} \bigcup_{k=1}^{N} \overset{\circ}{t}_{i,k} = \overset{\circ}{f}_i \\ (\forall\, 1 \leq k, k' \leq N) \quad k \neq k' \iff \overset{\circ}{t}_{i,k} \cap \overset{\circ}{t}_{i,k'} = \emptyset \end{cases}$$

where all the points of C are vertices of some of $t_{i,k}$ ($\overset{\circ}{p}$ denotes the interior of the polgyon p, in the sense of the natural topology induced on p by embedding it in $\mathbb{R}^2$). Therefore, none of the $t_{i,k}$ has an internal critical point (otherwise this point would belong to C, which is impossible because all points of C are vertices of some of $t_{i,k}$).

However, the restriction of Φ_e to some edges of this triangulation of f_i may have critical points[4]. Denote η such an edge. If the restriction of Φ to η has any nonisolated critical point, then the same argument as above holds and thus the corresponding edges do not need to be further refined. On the contrary, if such an edge critical point is isolated (in this case, the edge must be internal to f_i, as all isolated critical points along the edges of f_i have been inserted priorly), then Algorithm TRIANGULATE-BOUNDARIES recursively proceeds on η. However, the process terminates because all face critical points are supposed to be isolated, according to the hypothesis. Therefore, for each critical point p_i of the restriction of Φ_e to f_i, there exists a neighborhood $\mathcal{V}_i$ within which all directional derivatives of Φ_e are nonzero and thus, there exists a finite number of triangle subdivisions after which no edge critical points are left (because having such a critical point implies having one directional derivative equal to zero). □

An initial tetrahedralization of the interior is performed. When the finite element is starred into a set of tetrahedra, we know that the triangular base of each tetrahedron and its 3 bounding edges will not have any critical points since those have already been identified and inserted into the triangulation of the two-dimensional boundary of the element. However, the remaining 3 faces and 3 edges must be marked because Φ_e restricted to their domain may contain critical points. This is accomplished by MARK-TETRAHEDRON, which sets a bit code for each edge and face not on the base of the given tetrahedron (which must be properly oriented when passed to the subroutine).

[4] In other words, the subdivision of f_i cannot create new face critical points, but it can create new edge critical points.

```
TETRAHEDRALIZE-INTERIOR(M, T_0)
 1  S ← ∅
 2  T_1 ← ∅
 3  for e ← |M|
 4       do Let T ⊆ T_1 be all triangles on BDY^2(Ω_e)
 5          if |C_e| > 0
 6            then c ← C_{e,0}
 7            else  c ← CELL-CENTER(Ω_e)
 8          V ← STAR^3(c, T)
 9          for t ← V
10               do if MARK-TETRAHEDRON(t)
11                     then Push t onto S
12          for c ∈ {C_e \ C_{e,0}}
13               do Find t ∈ V s. t. c ∈ t
14                  Remove t from V and S
15                  U ← STAR^3(c, t)
16                  for t' ← U
17                       do if MARK-TETRAHEDRON(t')
18                             then Push t' onto S
19                                  Insert t' into V
20          Insert V into T_1
21  Return (T_1, S)
```

Now that we have an initial tessellation, we must search for critical points along the marked edges and triangles. Where these are found, they are inserted into the partition so that conditions (C1) and (C2) will not be violated. Note that CORRECT-TOPOLOGY will terminate when all critical points are isolated, but not when non-isolated critical points exist.

```
CORRECT-TOPOLOGY(M, S, T_1)
 1  while S not empty
 2      do Pop t from S
 3         C ← ∅
 4         for e ← marked edges of t
 5             do Insert c.p. of e into C
 6         for f ← marked faces of t
 7             do Insert c.p. of f into C
 8         for c ← C
 9             do Find t ∈ T_1 s. t. c ∈ t
10                Remove t from T_1 and S
11                U ← STAR^3(c, t)
12                for t' ← U
13                    do if MARK-TETRAHEDRON(t')
14                          then Push t' onto S
15                               Insert t' into T_1
```

The algorithms we have presented indicate that we must find critical points on arbitrary line segments and triangular faces in the domain of an element. Most polynomial system solvers require a power-basis representation of a system to be solved and that is not usually how finite elements are represented. Given that we wish to perform this change of basis as infrequently as possible, it behooves us to find a way to derive the restriction of Φ_e to a line or face from the full representation of Φ_e.

3 Feedback and Implementation Issues

As this research has progressed, we have discussed it with various groups at Sandia and tried to incorporate their feedback into our code. This section covers that feedback and the implementation issues affected. The feedback falls into two broad categories: that dealing with the polynomial basis functions and their numerical stability and solution, and that dealing with the ability of the mesh representation to handle different types of solvers (particularly, both those that share interpolant coefficients to achieve boundary continuity and those that do not, such as discontinuous Galerkin techniques).

3.1 Polynomial Interpolants

Working with higher order finite elements involves dealing with polynomial interpolants. So far, due to the time constraints of this research, we have only focused on tensor product Lagrange interpolants as they are the most commonly used elements in codes now in production at Sandia. Efficiently handling such interpolants involves limiting memory re-

quirements (so that most of the useful information can be retained in the cache memory) as well as being able to quickly evaluate them at any given point of the interpolation domain. In addition, as we need to find critical points, we also need to efficiently compute the partial derivatives of these interpolants. In this section, we explain how we addressed this problem by using algebraic techniques that allow to improve memory (and in particular cache) requirements, thus making our implementation faster. This involves a few theoretical results, that we are now describing. In all that follows, N will denote a natural number strictly greater than 1.

3.1.1 Definition and change of basis

The Lagrange interpolants associated to the given real numbers $\xi_0 < \cdots < \xi_N$ are the degree N polynomials of $\mathbb{R}[X]$ defined as follows:

$$(\forall i \in [\![0,N]\!]) \quad L_i^N(X) = \prod_{\substack{j=0 \\ j\neq i}}^{N} (X - \xi_j) \tag{3.1}$$

and their normalized versions are thus obtained *via*:

$$(\forall i \in [\![0,N]\!]) \quad \ell_i^N(X) = \frac{L_i^N(X)}{L_i^N(\xi_i)} = \prod_{\substack{j=0 \\ j\neq i}}^{N} \frac{X - \xi_j}{\xi_i - \xi_j}; \tag{3.2}$$

these take the value 1 at exactly one of the ξ_i, and vanish at all others. In our case, we use a uniform distribution of the Lagrange nodes over the parametric domain $[-1,1]$, and thus each ξ_i is simply $\frac{2i}{N} - 1$. Now, the factorized form (3.1) is not well-suited to the explicit calculation of the derivatives of these polynomials. Instead, we wish to express them in the power basis $\left\{X^n\right\}_{n\in[\![0,N]\!]}$, and thus explicit the coefficient such that

$$(\forall i \in [\![0,N]\!]) \quad L_i^N(X) = \sum_{n=0}^{N} a_n(i) X^n. \tag{3.3}$$

In fact, these coefficient can simply be retrieved using the elementary symmetric polynomials. For the sake of convenience, we introduce the following notation: for all $i \in [\![0,N]\!]$, Ξ_i^N denotes the N-tupled whose j^{th} component is ξ_j if $j < i$. ξ_{j+1} otherwise. In other words, it is the tupled formed by taking all ξ_j's, with the exception of ξ_i, and keeping them in the same order. Now, with this notation, it is a classical result that

$$(\forall (n,i) \in [\![0,N]\!]^2) \quad a_n(i) = (-1)^{N-n} S_{N-n}^N(\Xi_i^N), \tag{3.4}$$

where the S_n^N are the degree N elementary polynomials of $\mathbb{R}[X_1,\dots,X_N]$. As a reminder, here are a few specific cases:

$$\begin{aligned} S_1^N(X_1,\dots,X_N) &= \sum_{1\le i\le N} X_i && (3.5)\\ S_2^N(X_1,\dots,X_N) &= \sum_{1\le i<j\le N} X_iX_j && (3.6)\\ &\vdots && (3.7)\\ S_N^N(X_1,\dots,X_N) &= \prod_{1\le i\le N} X_i, && (3.8) \end{aligned}$$

or, in short form,

$$(\forall n\in[\![1,N]\!])\quad S_n^N(X_1,\dots,X_N)=\sum_{1\le i_1<\cdots<i_n\le N}\prod_{k=1}^{n}X_{i_k}. \tag{3.9}$$

We also set $S_0^N(X_1,\dots,X_N)=1$ to obtain (3.3) in its most generic form. To avoid the computing the symmetric polynomials and their derivatives at any point, it is therefore sufficient to precompute all necessary $a_n(i)$, and then use (3.3). If only one polynomial order is present within the entire mesh, then only $(N+1)^2$ coefficients need to be computed and statically stored, but in reality several polynomial orders are likely to be present in any real higher order finite element simulation; in fact, for a tensor-product 2-D or 3-D element, each coordinate's corresponding polynomial may very well have a different order. Therefore, it is interesting to notice that, thanks to the results that follows, we can divide the required storage size by 2.

Proposition 3.1. *Using the definitions above, if the ξ_i uniformly subdivide an interval that is symmetric about 0, then*

$$(\forall(n,i)\in[\![0,N]\!]^2)\quad a_n(N-i)=(-1)^{N-n}a_n(i). \tag{3.10}$$

Proof. If $n=N$, then the result is evident. Otherwise, we notice that the hypothesis implies, for all i in $[\![0,N]\!]$, $\xi_{N-i}=-\xi_i$. Therefore,

$$(\forall(n,i)\in[\![0,N-1]\!]\times[\![0,N]\!])\quad \Xi_{N-i}^N=-\sigma(\Xi_i^N), \tag{3.11}$$

where $\sigma\in\mathfrak{S}_N$ is the permutation over $N-$tuples that transposes each i^{th} entry with the $(N-i)^{\text{th}}$. On the other hand, (3.9) yields

$$\begin{aligned} (\forall n\in[\![1,N]\!])\quad S_n^N(-(X_1,\dots,X_N)) &= \sum_{1\le i_1<\cdots<i_n\le N}\prod_{k=1}^{n}-X_{i_k} && (3.12)\\ &= \sum_{1\le i_1<\cdots<i_n\le N}(-1)^n\prod_{k=1}^{n}X_{i_k} && (3.13)\\ &= (-1)^nS_n^N(X_1,\dots,X_N). && (3.14) \end{aligned}$$

Therefore, for any (n,i) in $[\![0,N-1]\!]\times[\![0,N]\!]$, we have,

$$S^N_{N-n}(\Xi^N_{N-i}) = S^N_{N-n}(-\sigma(\Xi^N_i)) = (-1)^{N-n}S^N_{N-n}(\sigma(\Xi^N_i)) = (-1)^{N-n}S^N_{N-n}(\Xi^N_i), \quad (3.15)$$

because S^N_{N-n} is indeed a symmetric polynomial, and thus is invariant under permutation of its variables. The conclusion then gently arises, by definition of a_n. □

3.1.2 Normalization factors

Given $N \in \mathbb{N}\setminus\{0,1\}$, define the following normalization coefficients:

$$(\forall i \in [\![0,N]\!]) \quad \mathcal{N}_N(i) = \frac{1}{L^N_i(\xi_i)} = \prod_{\substack{j=0\\ j\neq i}}^{N} \frac{1}{\xi_i - \xi_j}. \quad (3.16)$$

Using the fact that we use Lagrange interpolants over the parametric domain $[-1,1]$, we then have

$$(\forall i \in [\![0,N]\!]) \quad \frac{1}{\mathcal{N}_N(i)} = \left(\frac{2}{N}\right)^N \prod_{\substack{k=i-N\\ k\neq 0}}^{i} k = \left(\frac{2}{N}\right)^N (-1)^{N-i}(N-i)!\, i!. \quad (3.17)$$

from where it immediately follows that

$$(\forall i \in [\![0,N]\!]) \quad \mathcal{N}_N(N-i) = (-1)^N \mathcal{N}_N(i). \quad (3.18)$$

A practical consequence of (3.18) is that the storage space for the Lagrange normalization coefficients can be divided by either 2, when N is odd, or $2-\frac{2}{N+2}$, when N is even.

3.1.3 Issues

One of the failings of homotopy techniques is their requirement of input polynomials in the form of a power basis (also known as a monomial basis). The power basis can be numerically unstable because coefficients tend to vary by many orders of magnitude. Conversion from the Lagrange basis can leave these coefficients with very few significant digits. Also, when a polynomial is evaluated, terms with coefficients of opposite sign often nearly cancel each other out, leaving small quantities with very few significant digits of precision. The efficient computation of partial derivatives still requires a power basis representation, but other problems with homotopy continuation methods have led us to experiment with polynomial solvers that use resultants for univariate and bivariate polynomials [1]. These solvers are very fast but have trouble detecting repeated roots.

3.2 Efficient DOF storage and handling

As previously mentioned, finite element developers at Sandia have expressed interest in a visualization tool that is flexible enough to handle results from a wide variety of solvers; requests included

- a mesh representation that can accomodate discontinuous as well as continuous solutions,
- the ability to change the polynomial order of fields independently of each other, and
- the ability to change the polynomial order of individual faces and edges in the mesh without changing the cell as a whole.

These goals are accomplished by segregating coefficients of the polynomial basis functions by the regions of a finite element where the basis function vanishes. Each coefficient is called a DOF mode since it corresponds to a basis function mode shape. Groups of coefficients are called DOF nodes and are the basic unit of storage. DOF nodes may be shared between finite elements in the same way that corner points are typically shared. In this section, we discuss the difficulties of efficiently storing and fetching the degrees of freedom of higher order elements, and a solution using algebraic techniques. We focus on the case of continuous solutions since it is trivial to not share coefficients by simply storing them in a DOF node that is not referenced by more than one element.

3.2.1 Node Permutations

Each finite element cell stores a 32-bit integer containing a specification of the coordinate transform to move the shape functions from their storage order into the cell's order. Consider the example shown in Figure 5, where two cells, cell 0 and cell 1, share a face and thus refer to the same corresponding DOF node. This face is the 3$^{\text{rd}}$ in cell 0, while it the first in cell 1. Let's say that the face node has 3 DOF: $\{d,e,f\}$. Cell 1 uses these for interpolating a field F like so:

$$F = \ldots + dN_{2,1,2} + fN_{3,1,2} - eN_{2,1,3} + \ldots \tag{3.19}$$

with $N_{2,1,2} = \frac{1}{2}(1-s)\phi_2(r)\phi_2(t)$, $N_{3,1,2} = \frac{1}{2}(1-s)\phi_3(r)\phi_2(t)$, and so on. The shape functions are linear in s, indicating that the DOF correspond to a face mode in cell 1's $r-t$ plane (the figure shows $s=-1$). Note that e and f have been swapped and e has been negated. This implies that the storage order of the DOF is in reference to a different coordinate system than the cell.

We can generate a linear, 2×2, orthonormal transformation matrix to change from the DOF storage order into the cell's coordinates:

$$\begin{pmatrix} r_1 \\ t_1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{\text{store}} \\ t_{\text{store}} \end{pmatrix} \tag{3.20}$$

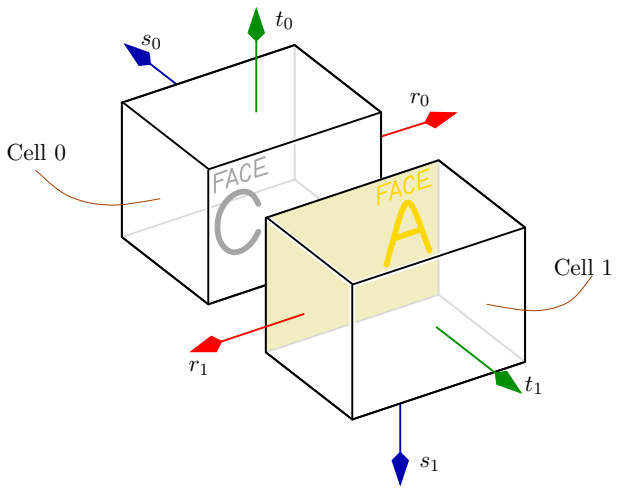

Figure 5. An example of the use of DOF node permutations stored with each cell.

where $r_{\text{store}}, t_{\text{store}}$ are the storage coordinates and r_1, r_1 are the cell coordinates. Think of this transformation as a change of basis from one coordinate system (storage) to another (cell 1).

$N_{2,1,2}$ will be unchanged by the transformation because $\phi_n(x)$ are symmetric about the origin for even n (*i.e.*, $n = 2i$). Shape functions with odd n (*i.e.*, $n = 2i+1$) are symmetric about $x = y$, so $\phi_{2i+1}(-x) = -\phi_{2i+1}(x)$. This property can be used to show that $eN_{3,1,2}(r_{\text{store}}) = -eN_{2,1,3}(t_1)$. Furthermore, given the transform, we can simply adjust the order and signs of the storage-order shape function coefficients before multiplying by cell-order shape functions to interpolate a value for F. This is a handy property because we can cache permuted coefficients from all DOF nodes for the entire cell and avoid the cost of applying transformations for each face and edge node each time F is evaluated.

In order to construct cell 2, we need to calculate the transformation between the storage coordinate frame of the shared face's DOF node and cell 2's coordinate frame. Unfortunately, there is no information on the coordinate frame stored with the DOF node – it is implicit. We're not lost, though, because we have a transformation from the storage coordinates to cell 1 and we can build a transformation from cell 1 to cell2.

The coordinate transform in the example above is one of 8 possible transforms for a quadrilateral face DOF node: there are 2 transforms associated with each corner vertex of the face. A hexahedron has 6 quadrilateral faces (3 bits each) and 12 edges (1 bit each), which means that all the permutations for a hexahedron (of any order) will fit into a 32-bit integer. Triangular faces have 6 possible permutations (still 3 bits), so wedge cells need 24 bits, pyramidal cells need 23 bits, tetrahedra need 18 bits. Octahedra need 36 bits, which is a bummer, but no one really uses those anyway, right? Each of the unique transformations for a hexahedron is assigned a 3 bit code. This code indicates which parametric coordinates

to swap and negate. The codes are illustrated in Figure 6. These codes can also be concatenated very easily. Thus, using the example above, we can assign a code to face A on cell 2 relative to face C on cell 1. Then, knowing the code for face C on cell 1, we can modify the code for face A in cell 2 such that it transforms the stored DOF node coefficients directly, again by appropriate swaps and negations.

Each cell has a coordinate frame that is specified by the ordering of the corner vertices in the connectivity array. All of the permutation bits stored with the cell are used to convert from each DOF node's storage frame into that *cell*'s coordinate system. It is important to realize that given a DOF node for face C on cell 1, the permutation bits do *not* specify a transformation from the storage frame into a coordinate frame associated with face C, but rather cell 1's coordinate frame. *All of the shape function coefficients are transformed from an arbitrary storage coordinate frame into the cell's coordinate frame.* Again, for hexahedral elements, the transformation amounts to swapping and negating selected coefficients.

3.2.2 Solution using the dihedral group D_4

The way we implemented the face DOF permutations is *via* the use of the dihedral group acting on a 4-element set, D_4. This provides a convenient abstract representation of the problem, and subsequently a straightforward way to implement these Lagrange-tensor face DOFs permutations. In fact, we first see these DOFs as 2-dimensional arrays, *i.e.*, matrices, and using D_4 we immediately obtain the index entries of any particular configuration shown in Figure 6 from the default configuration and then permutation bitcode. Finally, these 2-dimensional (matrix) indices are transformed into 1-dimensional (vector) indices, because memory storage is ultimately handled by vectors.

For any arbitrary $(d,m,n) \in \mathbb{N}^3$, we denote the DOF modes of a face as follows:

$$A = \begin{pmatrix} a_{n,0} & \cdots & a_{n,m} \\ \vdots & \ddots & \vdots \\ a_{0,0} & \cdots & a_{0,m} \end{pmatrix} \in \mathfrak{M}_{n+1,m+1}(\mathbb{R}^d),$$

where each $a_{i,j} \in \mathbb{R}^d$ represents the DOF modes of a Lagrange face interpolant (thus with order $(n+2)\times(m+2)$. Recall that the dihedral group D_4 is the group of permutations of 4 elements that is isomorphic the the group of isometries of the cube, *i.e.*,

$$D_4 = \big\{(), (0123), (0321), (02)(13), (01)(23), (03)(12), (02), (13)\big\}.$$

Evidently, we have $().A = A$, and the images of A by the other permutations of D_4 are given in the following paragraphs.

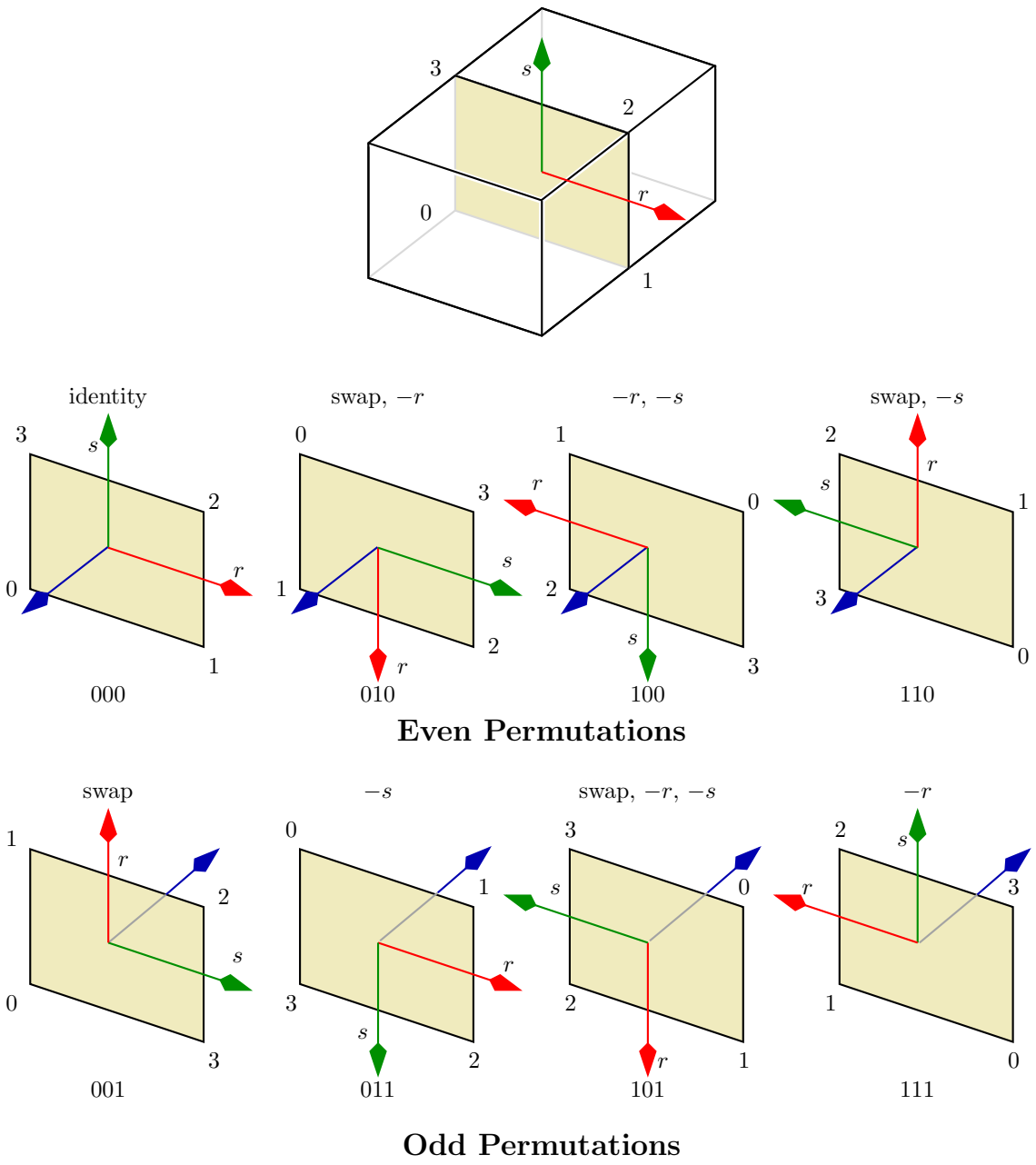

Figure 6. Codes for unique face node transformations for hexahedra.

Figure 7. A change of basis can be used to find the DOF node permutation for a new cell given an existing cell sharing that node.

Permuting with $(\mathbf{0123})$ Denoting $B = (0123).A$, we have $B \in \mathfrak{M}_{m+1,n+1}(\mathbb{R}^d)$, and

$$(0123).A = \begin{pmatrix} b_{m,0} & \cdots & b_{m,n} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,n} \end{pmatrix} = \begin{pmatrix} a_{0,0} & \cdots & a_{n,0} \\ \vdots & \ddots & \vdots \\ a_{0,m} & \cdots & a_{n,m} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,m]\!] \times [\![0,n]\!]\big) \quad \big((0123).A\big)_{i,j} = \big(A\big)_{j,m-i}$$

Permuting with $(\mathbf{0321})$ Denoting $B = (0321).A$, we have $B \in \mathfrak{M}_{m+1,n+1}(\mathbb{R}^d)$, and

$$(0321).A = \begin{pmatrix} b_{m,0} & \cdots & b_{m,n} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,n} \end{pmatrix} = \begin{pmatrix} a_{n,m} & \cdots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n,0} & \cdots & a_{0,0} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,m]\!] \times [\![0,n]\!]\big) \quad \big((0321).A\big)_{i,j} = \big(A\big)_{n-j,i}$$

Permuting with $(\mathbf{02})(\mathbf{13})$ Denoting $B = (02)(13).A$, we have $B \in \mathfrak{M}_{n+1,m+1}(\mathbb{R}^d)$, and

$$(02)(13).A = \begin{pmatrix} b_{n,0} & \cdots & b_{n,m} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,m} \end{pmatrix} = \begin{pmatrix} a_{0,m} & \cdots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n,m} & \cdots & a_{n,0} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,n]\!] \times [\![0,m]\!]\big) \quad \big((02)(13).A\big)_{i,j} = \big(A\big)_{n-i,m-j}$$

Permuting with $(\mathbf{01})(\mathbf{23})$ Denoting $B = (01)(23).A$, we have $B \in \mathfrak{M}_{n+1,m+1}(\mathbb{R}^d)$, and

$$(01)(23).A = \begin{pmatrix} b_{n,0} & \cdots & b_{n,m} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,m} \end{pmatrix} = \begin{pmatrix} a_{n,m} & \cdots & a_{n,0} \\ \vdots & \ddots & \vdots \\ a_{0,m} & \cdots & a_{0,0} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,n]\!] \times [\![0,m]\!]\big) \quad \big((01)(23).A\big)_{i,j} = \big(A\big)_{i,m-j}$$

Permuting with $(\mathbf{03})(\mathbf{12})$ Denoting $B = (03)(12).A$, we have $B \in \mathfrak{M}_{n+1,m+1}(\mathbb{R}^d)$, and

$$(03)(12).A = \begin{pmatrix} b_{n,0} & \cdots & b_{n,m} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,m} \end{pmatrix} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n,0} & \cdots & a_{n,m} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,n]\!] \times [\![0,m]\!]\big) \quad \big((03)(12).A\big)_{i,j} = \big(A\big)_{n-i,j}$$

Permuting with $(\mathbf{02})$ Denoting $B = (02).A$, we have $B \in \mathfrak{M}_{m+1,n+1}(\mathbb{R}^d)$, and

$$(02).A = \begin{pmatrix} b_{m,0} & \cdots & b_{m,n} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,n} \end{pmatrix} = \begin{pmatrix} a_{0,m} & \cdots & a_{n,m} \\ \vdots & \ddots & \vdots \\ a_{0,0} & \cdots & a_{n,0} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,m]\!] \times [\![0,n]\!]\big) \quad \big((02).A\big)_{i,j} = \big(A\big)_{j,i}$$

Permuting with $(\mathbf{13})$ Denoting $B = (13).A$, we have $B \in \mathfrak{M}_{m+1,n+1}(\mathbb{R}^d)$, and

$$(13).A = \begin{pmatrix} b_{m,0} & \cdots & b_{m,n} \\ \vdots & \ddots & \vdots \\ b_{0,0} & \cdots & b_{0,n} \end{pmatrix} = \begin{pmatrix} a_{n,0} & \cdots & a_{0,0} \\ \vdots & \ddots & \vdots \\ a_{n,m} & \cdots & a_{0,m} \end{pmatrix}$$

whence

$$\big(\forall (i,j) \in [\![0,m]\!] \times [\![0,n]\!]\big) \quad \big((13).A\big)_{i,j} = \big(A\big)_{n-j,m-i}$$

3.2.3 Implementation

In memory, the entries of A and its images are not stored as matrices, but as linear arrays. Therefore, the explicit formulas are as follows:

$$\big(\forall (i,j,k) \in [\![0,m]\!] \times [\![0,n]\!] \times [\![0,d-1]\!]\big)$$
$$\big((0123).A\big)[(i \times (n+1) + j) \times d + k] = A[(j \times (m+1) + m - i) \times d + k]$$

$$\big(\forall (i,j,k) \in [\![0,m]\!] \times [\![0,n]\!] \times [\![0,d-1]\!]\big)$$
$$\big((0321).A\big)[(i \times (n+1) + j) \times d + k] = A[((n-j) \times (m+1) + i) \times d + k]$$

$$\big(\forall (i,j,k) \in [\![0,n]\!] \times [\![0,m]\!] \times [\![0,d-1]\!]\big)$$
$$\big((02)(13).A\big)[(i \times (m+1) + j) \times d + k] = A[((n-i) \times (m+1) + m - j) \times d + k]$$

$$\big(\forall (i,j,k) \in [\![0,n]\!] \times [\![0,m]\!] \times [\![0,d-1]\!]\big)$$
$$\big((01)(23).A\big)[(i \times (m+1) + j) \times d + k] = A[(i \times (m+1) + m - j) \times d + k]$$

$$\big(\forall(i,j,k) \in [\![0,n]\!] \times [\![0,m]\!] \times [\![0,d-1]\!]\big)$$
$$\big((03)(12).A\big)[(i \times (m+1)+j) \times d+k] = A[((n-i) \times (m+1)+j) \times d+k]$$

$$\big(\forall(i,j,k) \in [\![0,m]\!] \times [\![0,n]\!] \times [\![0,d-1]\!]\big)$$
$$\big((02).A\big)[(i \times (n+1)+j) \times d+k] = A[(j \times (m+1)+i) \times d+k]$$

$$\big(\forall(i,j,k) \in [\![0,m]\!] \times [\![0,n]\!] \times [\![0,d-1]\!]\big)$$
$$\big((13).A\big)[(i \times (n+1)+j) \times d+k] = A[((n-j) \times (m+1)+m-i) \times d+k]$$

4 Conclusions

Since this proposal was funded for only 6 months, this annual report marks the end of the project. Our original goals were to

- correctly insert isolated critical points into a tessellation of finite elements that could be used for isosurfacing; and to
- study the implementation of higher order isosurfacing to detect non-isolated critical points.

Originally, we had hoped that the critical points of a scalar field over an open domain, along with critical points of the field's restriction to the domain's closure, result in a minimal but sufficient sampling of the finite element for analysis. However, we have had to modify this slightly to accomodate the fact that the restriction of the field to the closure of each 0-, 1-, and 2-simplex in the decomposition of the finite element must be recursively searched for critical points, not just the restriction to the boundaries of the finite element.

The algorithm and proof we present here accomplishes the tasks above with two exceptions:

- Non-isolated critical points whose locus is a curve or curved sheet in space will cause algorithm to fail to terminate. However, a simple limit on the number of iterations will yield a piecewise approximation of the locus of critical points. We do not see this as a problem.

- Non-isolated critical points that do not intersect a face or edge of any simplex in the decomposition of a finite element may not be detected. This will always be difficult because numerical methods frequently rely on the existence of a gradient to converge to a root; when the gradient does not exist in 1 or 2 directions, path-tracing root-finders have trouble maintaining a reasonable step size and other methods, such as the resultant techniques, must deal with singular matrices.

The implementation of these algorithms is nearly complete and our final report will include results.

In addition to the proposed work, we have studied a number of polynomial system solvers in an attempt to develop a more robust implementation. For univariate polynomials, the Lin-Bairstow solver is adequate – numerical accuracy is a concern because coefficient accuracy drops as roots are divided out of the result. For bivariate polynomials, the Day-Romero approach using Sylvester resultants may yield acceptable results, but is not sufficiently robust at locating repeated roots for use as a univariate solver. We have still not identified any technique that is uniformly robust for the three-dimensional case.

Apart from these minor issues, this paper has presented the algorithm and proof that were the main focus of this research and we have seen interest both at Sandia and outside in isocontouring higher-order finite elements. A conference paper on the implementation framework has been accepted and we have been invited to expand it into a journal article which will include some of the results of this research. The framework we present is flexible enough to represent finite element solutions from a large variety of simulation codes in both production and development at Sandia and we hope to continue developing it to include support for more polynomial bases and element shapes.

References

[1] D. Day and L. Romero. Roots of polynomials expressed in terms of orthogonal polynomials. *SINUM*, 43, 2005. Accepted/in Print.

[2] R. Khardekar and D. Thompson. Rendering higher order finite element surfaces in hardware. In *Proc. of Graphite 2003*, Melbourne, Australia, February 2003. ACM/SIGGRAPH.

[3] P. P. Pébay and D. C. Thompson. Parallel mesh refinement without communication. In *Proc. 13th International Meshing Roundtable*, Williamsburg, VA, September 2004.

[4] P. P. Pébay and D. C. Thompson. Communication-free streaming mesh refinement. *ASME Transactions, Journal of Computing & Information Science in Engineering, Special Issue on Mesh-Based Geometry*, 5(4), 2005. In press.

[5] W. J. Schroeder, F. Bertel, M. Malaterre, D. C. Thompson, P. P. Pébay, R. O'Bara, and S. Tendulkar. Framework for visualizing higher-order basis functions. In *IEEE Proc. Visualization 2005*, Minneapolis, MN, October 2005.

[6] P. Solin and L. Demkowicz. Goal-oriented hp-adaptivity for elliptic problems. *Comput. Methods Appl. Mech. Engrg.*, 193:449–468, 2004.

[7] D. Thompson and P. P. Pébay. Performance of a streaming mesh refinement algorithm. Sandia Report SAND2004-3858, Sandia National Laboratories, August 2004.

[8] D. C. Thompson, R. Crawford, R. Khardekar, and P. P. Pébay. Visualization of higher order finite elements. Sandia Report SAND2004-1617, Sandia National Laboratories, April 2004.

[9] D. C. Thompson and P. P. Pébay. Embarassingly parallel mesh refinement. *Engineering with Computers*, 2006. To appear, accepted August 2005.

DISTRIBUTION:

1 Pr Richard Crawford
Mech. Engr., C2200
The Univ. of Texas at Austin
Austin, TX 78703

1 Patricia Howard
Comp. & Appl. Mathematics
Rice University
6100 Main St. - MS 134
Houston, TX 77005-1892

1 Rahul Khardekar
1634 Oxford Street
Apt 305
Berkeley, CA 94709

2 Will Schroeder
Kitware
28 Corporate Drive
Suite 204
Clifton Park, NY 12065

1 Timothy J. Baker
Mechanical & Aerospace Engineering Department
Engineering Quadrangle
Princeton University
Princeton, NJ 08544

1 Pr Jérôme Pousin
I.N.S.A. Lyon
Center for Mathematics
Bâtiment Léonard de Vinci
69621 Villeurbanne cedex, France

1 Pr Pascal Frey
Laboratoire J-L Lions
Université Pierre et Marie Curie
Boîte courrier 187
75252 Paris cedex 05, France

1 MS 9051
Dawn Manley, 8351

4 MS 9051
Philippe P. Pébay, 8351

1 MS 9915
Mitchel W. Sukalski, 8961

1 MS 9217
Paul T. Boggs, 8962

1 MS 9217
Kevin R. Long, 8962

1 MS 9012
Jerry A. Friesen, 8963

1 MS 9012
Gary J. Templet, 8963

4 MS 9012
David C. Thompson, 8963

1 MS 0382
Kevin D. Copps, 9143

1 MS 1110
David M. Day, 9214

1 MS 1110
Louis Romero, 9214

1 MS 0822
Brian N. Wylie, 9227

1 MS 0822
Lee Ann Fisk, 9227

3 MS 9018
Central Technical Files, 8940-1

1 MS 0899
Technical Library, 9616

1 MS 9021
Classification Office, 8511, for
Technical Library, MS 0899, 9616
DOE/OSTI via URL